ISSN No:-2456-2165

Methods and Techniques for Recommender Systems in Secure Software Engineering: A Literature Review

Astrit Desku
Faculty of Contemporary Sciences and Technologies
South East European University
Tetovo, North Macedonia

Abstract:- Recommender Systems are software tools that can assist developers with a wide range of activities, from reusing codes to suggest developers what to do during development of these systems. All recommender systems should exert one or three of future functionalities: Gathering Data and Creating Dataset, Static Analysis and Recommendation to user-by-user interface. In this paper, we have presented a literature review in the field of recommender systems. Papers are aggregating by their context in three main groups: Mechanism to Collect Data, Recommendation Engine to Analyze Data and Generate Recommendations and User Interface to Deliver Recommendations. In the conclusion are presented number of reviewed paper for each category.

Keywords:- *Recommender Systems, Machine Learning, Software Engineering, Data Mining Techniques, Control Flow Graphs.*

I. INTRODUCTION

Software developers have always used tools to perform their work. In the earliest days of the discipline, the tools provided basic compilation and assembly functionality [1]. Then came tools and environments that increasingly provided sophisticated data about the software under development [1] [2]. Nowadays, the systematic and large-scale accumulation of software engineering data opened up new opportunities for the creation of tools or APIs (Application Programming Interfaces), that infer information estimated to be helpful to developers in a given context.

One way for helping developers during daily development activities is reusing code from previous project that they have developed or reuse code from projects developed from other developers. Today, there are online platforms (like Stack Overflow. GitHub etc.), in which developer posts their projects or snipped code from projects, these codes therefore can be used by other developers in their own projects. In most cases, these codes are used in worst fashion possible, copying from source and paste in their projects. This approach can be very risky reusing code in this way, considering that from 1.3 million Android applications that are analyzed 15.4 % contained security-related code snippets from Stack Overflow. Out of these 97.9 %, contain at least one insecure code snippet [1] [3].

Recommender Systems for Software Engineering (RSSEs) are software tools that can assist developers with a wide range of activities, from reusing codes to suggest developers what to do during development [5]. These tools can be used also to guide and recommend a developer for next activities, based on codes write from other developers. Many studies have analyzed the usability of security. Clark and Godspeed [4] and Whitten and Tygar [5] explore misuses of cryptography components from the end-user's point of view. Others analyze misuses of security APIs by application developers. It is worth noting that that over 83% of the vulnerabilities they analyzed from the CVE database were due to misuses of cryptography libraries while only 17% were caused by implementation bugs in the cryptography libraries themselves [4].

The aim of this paper is therefore to study the existing research in methods and techniques for recommender systems in software engineering, in order to analyze the state-of-art and to identify future directions. The method consists of the application of a systematic mapping study to extract as much literature as possible.

In order to identify related work, more than 150 papers selected have been classified by recommendation task and theirs sub categories and present a brief introduction to each paper or technique. At the end of each section a summary is presented regarding the gap identified in literature review, comparing with approach of using code type through a control-flow graph which latter can be used for generating a dataset for recommendation purposes.

II. RESULTS AND DISCUSSION

In modern software development, API are means to encapsulate responsibilities and facilitate code reuse and they are widely used in statically typed programming languages. Typical RSSE approaches apply static analyses to extract information about the usage of these APIs from the source code of many projects. Some RSSE techniques learn general patterns from source code by treating it as text or just on the syntax level [6].

In order to identify related work, we consulted a recent comprehensive survey and some recent publications renowned software engineering conferences. In the next sections, we will present paper recommenders grouped by their recommendation task and present a brief introduction to each paper or technique.

A. Mechanism to Collect Data

While for researchers in different field of studies, data is very important part of their research, for researchers in recommender systems finding reliable data or dataset is one of the biggest challenges. Nowadays versioned source code is available in many public repositories of open-source projects, it is much harder to get access to more detailed change information or to activities that describe the in-IDE (Integrated Developer Environment) development process.

Once the important properties of a target domain have been identified, extensive datasets can be generated through simulation [7]. Unfortunately, the in-IDE development process is very complex and not yet fully understood to be able to simulate it. As a result, it is required to create appropriate datasets.

Based on literature review for this work datasets for recommender systems can be grouped in three main categories. In the following of this section, we will present these datasets and a short description for each of them specifically.

a) Source Code
Several approaches make source code available for research through in large and stable datasets. The PROMISE repository presented in [8] was an early advance to provide a platform to share datasets and tools used in mining studies.

The QUALITASCORPUS [9] is a curated collection of open-source JAVA source code for empirical research. The corpus contains dependencies for most contained projects and describes how the missing dependencies can be installed.

In [10] authors extract a dataset for usages of API methods and annotations from a large number of GitHub repositories. From all libraries that have been referenced in the repositories, the authors select the five most popular open-source projects that are reasonably large and actively developed.

The BOA project, presented firstly in [11] provides an infrastructure for writing analyses with an ultra-large-scale repository. The project provides curated data sets that contain source code from many projects hosted on GitHub (about 7.8M projects) and Source Forge (about 700K projects). The source code is stored in a custom AST format that supports most JAVA constructs.

b) Source Code under Development
Changes extracted from the commit history of repositories are coarse-grained and not representative for actual source code evolution [12].

Other means are required to capture intermediate states of source code to be able to find and analyze the problems programmers are facing during development. Several tools exist that can record source code that change during regular coding activities to enable studies of the evolution of source code [13].

In [14], the authors reports that 80% of these snapshots can be compiled, these works usually capture immediate states on save, loosing many intermediate edit steps in between. A different form of incomplete source-code that is still under development can be found on question and answer sites like Stack Overflow or REDDIT. These sites have become an important data source for empirical research on software engineering. In [15] it is presented a solution with an island grammar that can be used to parse Stack Overflow posts into heterogeneous AST (H-AST). The grammar supports JAVA, XML, JSON, stack traces, and text fragments and can be used to transform released Stack Overflow data dumps.

c) Meta Data
Another kind of dataset exists that does not focus on source code, but on the meta data that describes projects. In the context of this work, these datasets are interesting as an additional data source that can be integrated into analyses to enrich existing data.

In [16] was presented OSSMOLE, which was one of the first advances to create a high-quality database of FLOSS project information for research. The authors achieve this through publishing standard analyses that enable replication of results and through facilitating reuse of analysis scripts by others. GHTorrent presented in [17] is an effort to make the vast amount of development activities on GitHub available for research. The project stores the development events of GitHub repositories, for example,activities that include push, fork, or branch operations.

While openHUB is a website that curates metadata for a large number of open-source projects. The provided data consists of general meta-data such are repository URLs, main programming language, and license, and of several metrics regarding the source code, the activities of contributors, and historical data regarding the evolution of the project. The database can be accessed through a REST-based API.

d) Interactions
One of the earliest and most extensive datasets of developer interactions is the public dataset of the ECLIPSE USAGE DATA COLLECTOR. The intention of this project was to provide a means for plugin developers to analyze which functionality of ECLIPSE was actually being used by their users. To this end, the dataset contains a log of executed commands and activated windows, grouped by user [18]. In [19] it was presented a recommender system called RASCAL. It is a recommender agent that tracks usage histories of a group of developers to recommend to an individual developer components that are expected to be needed by that developer. Further they introduce a content-based filtering technique for ordering the set of recommended software components and present a comparative analysis of applying this technique to a number of collaborative filtering algorithms.

In [20] was presented the BLAZE tool that can be installed in VISUAL STUDIO 2010 to track which commands are invoked by the developer in the IDE. In [21] authors declare that they deployed it within ABB INC. and tracked activities of almost 200 developers, covering more than 30K hours of active development time.

In [22] was proposed DFLOW to capture information about in-IDE activities. To this end, they capture source-code changes on a structural level (method rename), windows that are being interacted with, and the layout space occupied by open windows. Their dataset covers 750 hours of development work of 17 developers.

e) Section Discussion
In the papers presented above, different aspects and approaches to data collection mechanisms have been highlighted, which are grouped into several categories depending on the collection methods. The problems with above mentioned approaches are that many of them focus on collecting metadata about codes from various repositories. In this aspect as a gap that can be identified is lack of automatic suggestion from the code structure itself comparing with approach delve deeper in the code structure by analyzing and representing the code type through a control-flow graph which latter can be used for generating a dataset for recommendation purposes.

B. Recommendation Engine to Analyze Data and Generate Recommendations

Recommendation engine and data analyses or in literature can be found also as static analyses is the most important part of recommender systems. There are a lot of studies that treat this part, in the following will be some of them based on literature reviewed.

In [23] it is built a call recommendation, which was later extended in [18]. The underlying approach in both cases identifies object instances in an intra-class analysis and extracts all method invocations on each instance, as well as a description of the surrounding source code. In [24] was proposed a recommender system called Precise, which focuses on parameter call sites. They extract several features from the structural context that describe the method invocation and its parameters: the called method, the enclosing method, methods that are called on the receiver, and methods that are called on the parameter. In [25] mainly use identifiers to predict method calls. They extract all identifiers used in the source code and split names on camel-case humps. Even though control structures are not part of their model, they consider control structure keywords in the tokenization. They apply text-mining techniques such as stemming or removing stop words to unify the collected tokens.

In [26] it was implemented a tool that learns correct API usage from interactions of developers in their IDE. When code completion is triggered in the IDE, they extract features from the structural context around the trigger point that include the type, definition, enclosing statement, expression type, enclosing method.

In [27] they solve the task of call recommendation by mapping it to a text-mining problem. They model sequences of methods calls as sentences. Their main idea is to reduce the problem of code completion to a natural-language processing problem of predicting probabilities of sentences. They designed a simple and scalable static analysis that extracts sequences of method calls from a large codebase, and index these into a statistical language model. We then employ the language model to find the highest ranked sentences, and use them to synthesize a code completion.

In [28] it was presented Context Sensitive Code Completion - CSCC system that is built on a simple and efficient algorithm. The approach tokenizes source code by traversing an AST (Abstract Syntax Trees). Tokens will be created for JAVA keywords, types, and, method names. CSCC is context sensitive in that it uses new sources of information as the context of a target method call. CSCC indexes method calls in code examples by their contexts. To recommend completion proposals, CSCC ranks candidate methods by the similarities between their contexts and the context of the target call.

In [29] is proposed CODEWEB, a tool that can be used to identify “reuse relationships” in source code. The developer can consult these tuples to learn about the correct API usage. In [30] is present JAVA RULE FINDER, a tool that infers rules about a correct usage of a framework directly from its source code. The rules are prepared in a textual format and serve as a browsable documentation for developers. In [31] authors propose a system that automatically generates the documentation of an API method. Instead of analyzing the implemented behavior in the method, they look at callers of the method and extract descriptive information from there. In [32] is proposed PROMPTER, a tool that proposes relevant postings on STACKOVERFLOW to developers while they are working in the IDE. It extracts the current programming context from the source code under edit. The context contains fully qualified names for types and methods, the source code of the enclosing element at the edit location, and a list of all types and methods used in the enclosing element that are defined outside of the project. In the [33], authors have presented a software product line called OpenCCE, which in reality is an Eclipse plugin. This solution combines the advantages of documentation and program analysis with ease of use and availability. In fact, solution separates API users from the domain knowledge required to understand these APIs through an expert system.

Anomaly detection approaches learn characteristics of a typical correct program. The underlying models are then used to detect deviations from this established norm. Based in literature review in the following are presented some approach with defect and anomaly detection.

In [34] authors propose DMMC to detect missing method calls. They extract object usages that describe how an object instance is used. They encode the type of the object, the enclosing method, and all calls on it. In [35] authors present PR-MINER, another detector for missing method calls. The tool extracts facts from a method such as the type of variable declarations, variable names, assignments, and calls and uses a prefixing strategy to prevent name collisions in different scoping levels. In [36] authors present an approach to build finite-state-automatons that describe valid protocols for using a specific type. The approach is based on their own framework that can be used to mine method sequences [37]. In [38] authors present a data-driven approach to vulnerability detection using machine learning, specifically applied to C and C++ programs. During the build process, they extract features at

ISSN No:-2456-2165

two levels of granularity. At the function level, they extract the Control Fow Graph (CFG) of the function. Within the control flow graph, they extract features about the operations happening in each basic block and the definition and use of variables.

a) Section Discussion
Many of the overhead analyzed papers focus their attempts on building recommendation from simplistic code analysis. The focus is given either to the perspective of method invocations used in the codes subject of the analysis, either they imply a direct natural language processing techniques in order to gain insights from the code and latter perform some machine learning technique for generating recommendations or they extract limited features from the code, such as functions to generate the control-flow-graph (CFG) from the code for recommendation purposes. The above-mentioned papers analyzed failed to address the wholistic approach towards code analysis, i.e. the programming codes are often too complex to be analyzed from a single aspect alone. The goal of our approach is to use a more holistic methods towards code analysis by using Control Flow Graph techniques for code pattern analysis. In this way, a system will recommend a more effective and appropriate action for software developers to automatically ensure that their software is more secure.

C. User Interface to Deliver Recommendations

User interface is the part by which users interact with recommender systems. Interaction can be implemented in different ways, some solution are implemented via API or plug-in, which are integrated in existing development tools like Eclipse or Visual Studio, otherwise some solutions are integrated solutions which includes gathering data, static analyses and delivering recommendations to users.

In the following will be presents some solutions that are grouped in two groups based on solution approach.

a) Guiding Software Changes
In many web sites that we use for buying a book or smoothening else, in time when we chose to, we can encounter recommendations of the form, “Customers who bought this also bought….” Such suggestions stem from purchase history. Buying two or more things together establishes a relationship between them, which web sites uses to create recommendations. In [39], they present the eRose plug-in for the Eclipse integrated development environment (IDE) that realizes a similar feature for software development by mining past changes from version archives. This feature tracks changed elements and updates recommendations in a view after every save operation. For example, if a developer wants to add a new preference to the Eclipse IDE and so changes fKeys[] and initDefaults(), eRose would recommend “Change plugin.properties” because all developers who changed the Eclipse code did so in the past. In [40], a system called GROUMINER was developed to find patterns in API usages from source code repositories. Their approach is based on the syntax tree; they do not consider resolved types. The patterns, called Groums, contain ordered information about method calls, object declarations, and control points. In their follow-up work, in [41] they propose the GRAPACC snippet recommender that uses these patterns. PROSPECTOR presented in [42] and PARSEWEB presented in [43] are two recommenders that propose ordered sequences that involve different API types. Both recommenders suggest call sequences that show the developer how to get from one API type to another.

b) Code Search
Code search is quite similar to snippet recommendations. The difference is that proposals point to existing examples that were observed in repositories or the local workspace, instead of making probabilistic recommendations. The proposals cannot be directly integrated into the current editor, but it will point to source code that can be used by the developer to understand a correct usage of the API in question.

In [44] the authors present MAPO, a code search tool that mines method sequences from API usage examples. Methods calls include constructor calls, static and non-static calls, as well as casts. In [45] the authors present the code search tool STRATHCONA for the ECLIPSE IDE. The tool retrieves relevant source code examples to help developers use frameworks effectively. When a developer wants to find how to do any specific code can highlight the par¬tially complete code, the context and ask Strathcona for similar examples. In [46], system called MUSE was proposed. It represent a tool that provides developers with examples of how to use a particular method. The tool requires the source code of the target API as well as a list of its clients. Whenever a call to one of the target API public methods is observed in the client code, an intra-procedural backward slice is created that shows how to get to this call. In [47] present CODECONJURER that supports developers by searching for working code that follows a specified UML-like syntax. The involved static analysis is based on the structural context only.

c) Section Discussion
In almost all of the above-mentioned papers, there is a lack of attention given to Control Flow Graphs (CFG) for code pattern extraction. CFGs are a well founded and described in software testing. However, as the literature review conducted by the authors suggest that a very little attention has given to CFGs for code pattern analysis that can be used for generating datasets which furthermore can be utilized in Machine Learning for code recommendation. As a result, this research path is worth exploring.

ISSN No:-2456-2165

III. CONCLUSION

Based on the most literature consulted during this work, the recommendation systems for software engineering (RSSEs) are defined as a software tools introduced specifically to help software development teams and stakeholders to deal with information seeking and decision-making.

They comprise three main components, which are [2]:

- Mechanism to collect data,
- Recommendation engine to analyze data and generate recommendations,
- User interface to deliver recommendations.

In this work we have presented past work from different researchers in the field of recommender systems, different approaches to the use of data mining algorithms for static analysis and different tools that are developed as outcome of these research and which now are using by developers during coding processes.

Based in literature review, in the table below are presented publication grouped by recommendation task and theirs sub categories.

RecommendationTask	Categories	Publication
Mechanism to Collect Data	Source Code	[8],[9], [10], [11]
	Source Code Under Development	[12], [13], [14], [15]
	Meta Data	[16], [17]
	Interactions	[18], [19], [20], [21], [22]
Recommendation Engine to Analyze Data and Generate Recommendations		[23], [18], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [48]
User Interface to Deliver Recommendations	Guiding Software Changes	[39], [40],[41], [42], [43]
	Code Search	[44], [45],[2], [46], [47]

Table 1: Publication grouped by recommendation task

REFERENCES

[1.] M. P. Robillard, W. Maalej, R. J. Walker and T. Zimmermann, Recommendation Systems in Software Engineering, Heidelberg New York Dordrecht London: Springer-Verlag, 2014.

[2.] M. P. Robillard, R. J. Walker and T. Zimmermann, "Development tools: Recommendation Systems for Software Engineering," I E E E S O F T WA R E www. c omp u t e r . o r g / s o f tw a r e.

[3.] F. Fischer, K. B¨ottinger, H. Xiao, C. Stransky, Y. Acar, M. Backe and S. Fahl, "Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security," IEEE Symposium on Security and Privacy, 2017.

[4.] S. Clark, T. Goodspeed, P. Metzger, Z. Wasserman, K. Xu and M. Blaze , "Why (special agent) Johnny (still) can't encrypt: a security analysis of the APCO project 25 two-way radio system," in Proceedings of the 20th USENIX conference on Security , Berkeley, 2011.

[5.] Whitten and J. D. Tygar, "Why Johnny can't encrypt: a usability evaluation of PGP 5.0," in Proceedings of the 8th conference on USENIX Security Symposium - Volume 8 , Washington, 1999.

[6.] R. Holmes and G. C. Murphy, "Using Structural Context to Recommend Source Code Examples," in International Conference on Software Engineering (ICSE), 2005.

[7.] L. Pickard, B. Kitchenham and a. S. Linkman, "An investigation of analysis techniques for software datasets," in nternational Software Metrics Symposium, 1999.

[8.] S. Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering," http://promise.site.uottawa.ca/SERepository, University of Ottawa, Canada, 2005.

[9.] Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton and J. Noble, "Qualitas corpus: A curated collection of java code for empirical studies," in Asia Pacific Software Engineering Conference, 2010.

[10.] A. Sawant and A. Bacchelli, "A dataset for API usage," in International Conference on Mining Software Repositories, 2015.

[11.] R. Dyer, H. A. Nguyen, H. Rajan and T. N. Nguyen, "BOA: Ultra-Large-Scale Software Repository and Source-Code Mining.," Transactions on Software Engineering and Methodology, 2015.

[12.] S. Negara, M. Vakilian, N. Chen, R. E. Johnson and D. Dig, "Is it dangerous to use version control histories to study source code evolution?," in European Conference on Object-Oriented Programming, Springer, 2012.

[13.] R. Robbes and M. Lanza, "SpyWare: A Change-aware Development Toolset," in International Conference on Software Engineering, ICSE, NY, 2008.

[14.] J. Spacco, J. Strecker, D. Hovemeyer and W. Pugh, "Software Repository Mining with Marmoset: An Automated Programming Project Snapshot and Testing System," in International Workshop on Mining Software Repositories, 2005.

[15.] L. Ponzanelli, A. Mocci and M. Lanza, "StORMeD: Stack Overflow Ready Made Data," in International Conference on Mining Software Repositories, 2015.
[16.] M. Conklin, J. Howison and K. Crowston, "Collaboration Using OSSmole: A Repository of FLOSS Data and Analyses," in International Workshop on Mining Software Repositories, AMC, 2005.
[17.] G. Gousios, "The ghtorrent dataset and tool suite," in Working Conference on Mining Software Repositories, NY, 2013.
[18.] S. Proksch, L. Johanes and M. Mezini, "Intelligent Code Completion with Bayesian Networks," ACM Transactions on Software Engineering and Methodology (TOSEM), p. Article No. 3 , 2015.
[19.] Mccarey, M. Ó. Cinnéide and a. N. K. Rascal, "A Recommender Agent for Agile Reuse," Artificial Intelligence Review, p. 253–276, 2005.
[20.] W. Snipes, A. R. Nair and E. Murphy-Hill, "Experiences Gamifying Developer Adoption of Practices and Tools," in International Conference on Software Engineering, 2014.
[21.] Singh, L. L. Pollock, W. Snipes and N. A. Kraft, "A case study of program comprehension effort and technical debt estimations," in International Conference on Program Comprehension, 2016.
[22.] R. Minelli, A. Mocci, R. Robbes and M. Lanza, "Taming the ide with fine-grained interaction data," in International Conference on Program Comprehension, 2016.
[23.] M. Bruch, M. Monperrus and M. Mezini, "Learning from Examples to Improve Code Completion Systems," in International Symposium on the Foundations of Software, 2009.
[24.] C. Zhang, J. Yang, Y. Zhang, J. Fan, X. Zhang, J. Zhao and P. Ou, "Automatic Parameter Recommendation for Practical API Usage," in International Conference on Software Engineering, IEEE, 2012.
[25.] L. Heinemann, V. Bauer, M. Herrmannsdoerfer and B. Hummel, "Identifier-based Context-dependent API Method Recommendation," in European Conference on Software Maintenance and Reengineering, IEEE, 2012.
[26.] S. Amann, S. Proksch and M. Mezini, "Method-call Recommendations from Implicit Developer Feedback," in International Workshop on CrowdSourcing in Software Engineering, 2014.
[27.] Raychev, M. Vechev and E. Yahav, "Code Completion with Statistical Language Models," in Conference on Programming Language Design and Implementation, 2014.
[28.] M. Asaduzzaman, C. K. Roy, K. A. Schneider and D. Hou, "CSCC: Simple, Efficient, Context Sensitive Code Completion," in ICSME, 2014.
[29.] Michail, "Data Mining Library Reuse Patterns in User-selected Applications," International Conference on Automated Software Engineering. IEEE, 1999.
[30.] H. Zhong, L. Zhang and H. Mei, "Inferring Specifications of Object-oriented APIs from API Source Code," Asia-Pacific Software Engineering Conference, 2008.
[31.] P. W. McBurney and C. McMillan, "Automatic Documentation Generation via Source Code Summarization of Method Context," International Conference on Program Comprehension., 2014.
[32.] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto and M. Lanza, "Mining StackOverflow to Turn the IDE Into a Self-Confident Programming Prompter," International Conference on Mining Software Repositories, 2014.
[33.] S. Arzt, K. Ali, S. Nadi, E. Bodden, S. Erdweg and M. Mezini, "Towards Secure Integration of Cryptographic Software," ACM International Symposium on New Ideas , pp. 1-13, 2015.
[34.] M. Monperrus, M. Bruch and M. Mezini, "Detecting Missing Method Calls in Object Oriented Software," in European Conference on Object-oriented Programming, 2010.
[35.] Z. Li and Y. Zhou, "PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code," in International Symposium on Foundations of Software Engineering, 2005.
[36.] M. Pradel, C. Jaspan, J. Aldrich and T. R. Gross, "Statically Checking API Protocol Conformance with Mined Multi-object Specifications," in International Conference on Software Engineering, 2012.
[37.] M. Pradel, P. Bichsel and a. T. Gross, "A Framework for the Evaluation of Specification Miners Based on Finite State Machines," in International Conference on Software Maintenance, 2010.
[38.] J. A. Harer, L. Kim, R. L. Russell, O. Ozdemir, O. Ozdemir, E. Antelman and S. Chin, "Automated software vulnerability detection with machine learning," in ResearchGate, 2018.
[39.] T. Zimmermann, A. Zeller, P. Weissgerber and S. Diehl, "Mining version histories to guide software changes," IEEE Transactions on Software Engineering, vol. 31, no. 6, pp. 429 - 445, 2005.
[40.] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi and T. N. Nguyen, "Graph-based Mining of Multiple Object Usage Patterns," in International Symposium on Foundations of Software Engineering, 2009.
[41.] T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. A. Kofahi and T. N. Nguyen, "Graph-based Pattern-oriented, Context-sensitive Source Code Completion," in International Conference on Software Engineering, 2012.
[42.] D. Mandelin, L. Xu, R. Bodík and D. Kimelman, "Jungloid Mining: Helping to Navigate the API Jungle," in Programming Language Design and Implementation, 2005.
[43.] S. Thummalapenta and T. Xie, "Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web," in International Conference on Automated Software Engineering, 2007.
[44.] H. Zhong, T. Xie, L. Zhang, J. Pei and H. Mei, "MAPO: Mining and Recommending API Usage Patterns," in European Conference on Object-oriented Programming, 2009.
[45.] R. Holmes, R. J. Walker and G. C. Murphy, "Approximate Structural Context Matching: An Approach to Recommend Relevant Examples.," in Transactions on Software Engineering, 2006.
[46.] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto and A. Marcus, "How Can I Use This Method?," in

nternational Conference on Software Engineering, 2015.
[47.] O. Hummel, W. Janjic and C. Atkinson, "Code Conjurer: Pulling Reusable Software out of Thin Air," IEEE, 2008.
[48.] V. Phan, M. L. Nguyen and L. T. B, "Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction," 2018.
[49.] M. Robillard, R. Walker and T. Zimmermann, "“Foreword,” Proc. Int’l Workshop on Recommendation Systems for Software Engineering," in ACM Press, 2008.
[50.] M. Robillard, R. Walker and T. Zimmermann, "Recommendation Systems for Software Engineering," IEEE Software, pp. 80-86, 2010.
[51.] H.-J. Happel and W. Maalej, "Potentials and Challenges of Recommendation Systems for Software Development," in ACM, 2008.
[52.] Spillner, T. Linz and H. Schaefer, Software Testing Foundations: A Study Guide for the Certified Tester Exam, Rocky Nook, 2007.
[53.] Hambling, P. Morgan, A. Samaroo, G. Thompson and P. Williams, SOFTWARE TESTING, British Informatics Society Limited, 2010.
[54.] J. B. Schafer, "The Application of Data-Mining to Recommender Systems," in U n i v e r s i t y o f N o r t h e r n I o w a.
[55.] S. Proksch, S. Amann, S. Nadi and M. Mezini, "Identifying Requirements of Static Analyses for Recommendation Systems in Software Engineering".
[56.] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini and T. Ratchford, "Automated API Property Inference Techniques," IEEE Transactions on Software Engineering, vol. 39, no. 5, pp. 613-637, 2013.
[57.] M. K. N. Mahrin, A. H. Mohamed and M. Naz’ri, "A survey on data mining techniques in recommender systems," in Springer-Verlag part of Springer Nature, Germany, 2017.
[58.] D. Nembhard, M. M. Carvalho and T. C. Eskridge, "Towards the application of recommender systems to secure coding," in EURASIP Journal on Information Security, 2019.
[59.] Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," IEEE Transactions on Knowledge and Data Engineering, vol. 17 , no. 6, pp. 734 - 749, June 2005.
[60.] M. Bruch, T. Schafer and M. Mezini, "FrUiT : IDE support for framework understanding," in Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange, New York, 2006.
[61.] M. Bruch and M. Mezini, "Improving code recommender systems using Boolean factor analysis and graphical models," in Proceedings of the International Workshop on Recommendation Systems for Software Engineering (RSSE'08), New York, 2008.
[62.] M. Bruch, M. Monperrus and M. Mezini, "Learning from examples to improve code completion systems," in In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, New York, 2009.
[63.] T. Gvero, V. Kuncak, I. Kuraj and R. Piskac, "Complete Completion Using Types and Weights," in Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, New York, 2013.
[64.] Hindle , E. T. Barr, Z. Su, M. Gabe and P. Devanbu, "On the Naturalness of Software," in Proceedings of the 2012 International Conference on Software Engineering, NJ, 2012.
[65.] D. Jackson, Software Abstractions: logic, language, and analysis, Cambridge: MIT Press, 2012.
[66.] Kaur, G. Singh and J. Minhas, "A Review of Machine Learning based Anomaly," International Journal of Computer Applications Technology and Research, pp. 185 - 187, 2013.
[67.] D. Lazar, H. Chen, X. Wang and N. Zeldovich, "Why does cryptographic software fail?: a case study and open problems," in Proceedings of 5th Asia-Pacific Workshop on Systems, New York, 2014.
[68.] K. Lewin, "Action research and minority problems," Journal of social issues , pp. 34-46, 1946.
[69.] Z. Li and Y. Zhou, "Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code," in Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, New York, 2005.
[70.] Michail, "Data mining library reuse patterns using generalized association rules," in Proceedings of the 22Nd International Conference on Software Engineering, New York, 2000.
[71.] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini and T. Ratchford, "Automated API Property Inference Techniques," IEEE Transactions on Software Engineering, pp. 613-637, 2013.
[72.] M. Robillard and R. DeLine, "A field study of API learning obstacles," Empirical Software Engineering, pp. 703-732, 2011.
[73.] S. Rose, N. Spinks and A. I. Canhoto, Management Research, Abingdon, New York: Routledge, 2015.
[74.] Jyothsna, V. V. R. Prasad and K. M. Prasad, "A review of anomaly based intrusion detection systems," International Journal of Computer Applications, pp. 26-35, 2011.
[75.] Eckhardt, "Various aspects of user preference learning and recommender systems," 2013.
[76.] X. Amatriain, A. Jaimes, N. Oliver and J. M. Pujol, "Data Mining Methods for Recommender Systems," Recommender Systems Handbook, no. Springer, Boston, MA, pp. 39-71, 2011.
[77.] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in 20th International Conference on Very Large Data Bases, 1994.